Национальный исследовательский ядерный университет «МИФИ»

(Московский Инженерно-Физический Институт)

Кафедра №42 «Криптология и кибербезопасность»

Лабораторная работа №3

«Реализация алгоритма с использованием технологии OpenMP»

Тимин Александр Б21-515 (2023г.)

Рабочая среда:

- Процессор: AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz, 8 ядер (16 логических)
- Оперативная память: 16.0 GB DDR4 3200 МГц
- ОС: Windows 10 Pro 22H2 64-bit operating system, x64-based processor
- Среда разработки: Microsoft Visual Studio 2022 (v143)
- Версия OpenMP: 200203 (/openmp:llvm)

Последовательный алгоритм

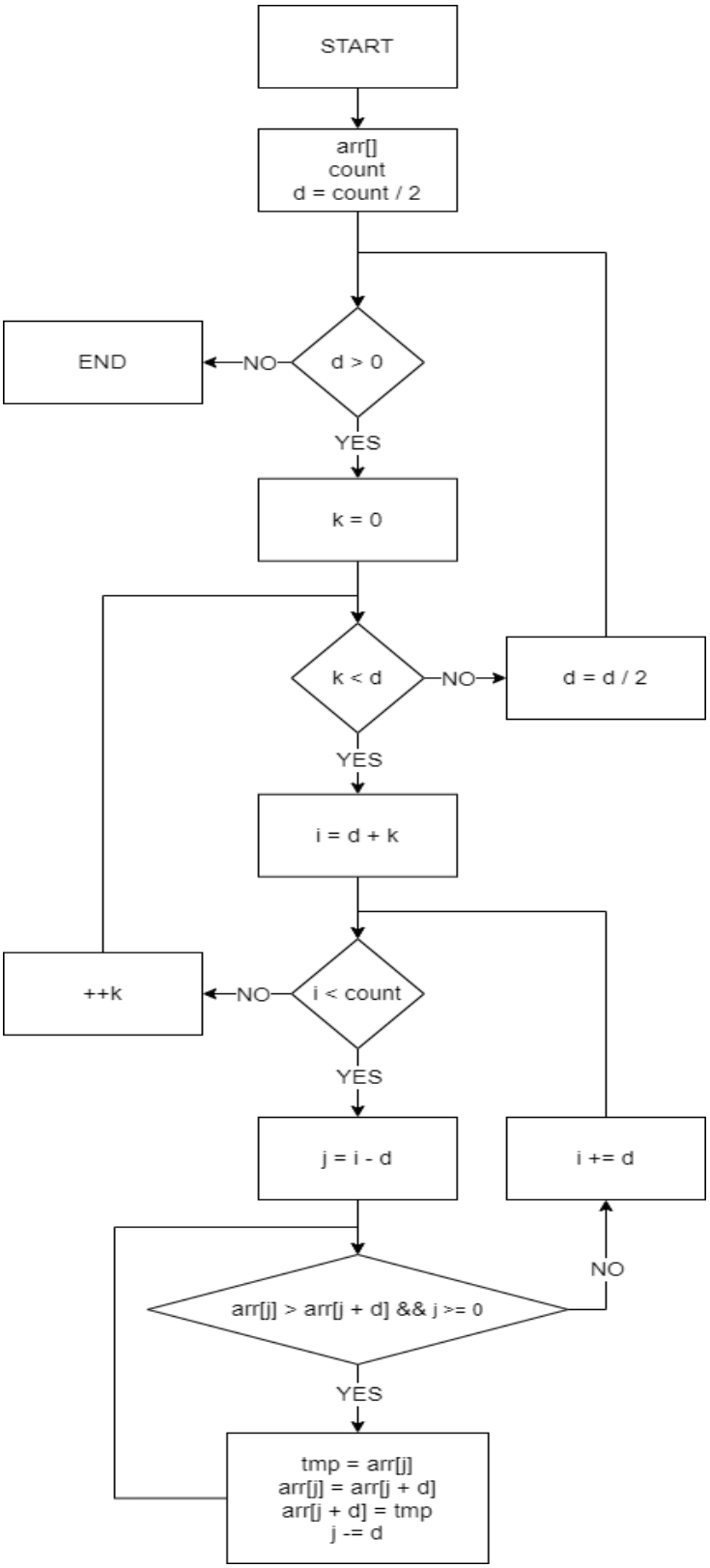
Алгоритм сортировки Шелла основывается на алгоритме сортировки вставками. Выбирается убывающая последовательность натуральных чисел (последнее – строго 1). Для каждого числа этой последовательности – d: массив условно разбивается на несколько массивов так, чтобы соседние числа в новом массиве отстояли друг от друга на d (получаем d новых массивов); каждый такой новый массив сортируется по алгоритму сортировки вставками. В данной работе последовательность чисел d состоит из count/2, count/4, ..., 1, где count – размер исходного массива.

Алгоритм взят с сайта:
https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%A8%D0%B5%D0%BB%D0%BB%D0%B0

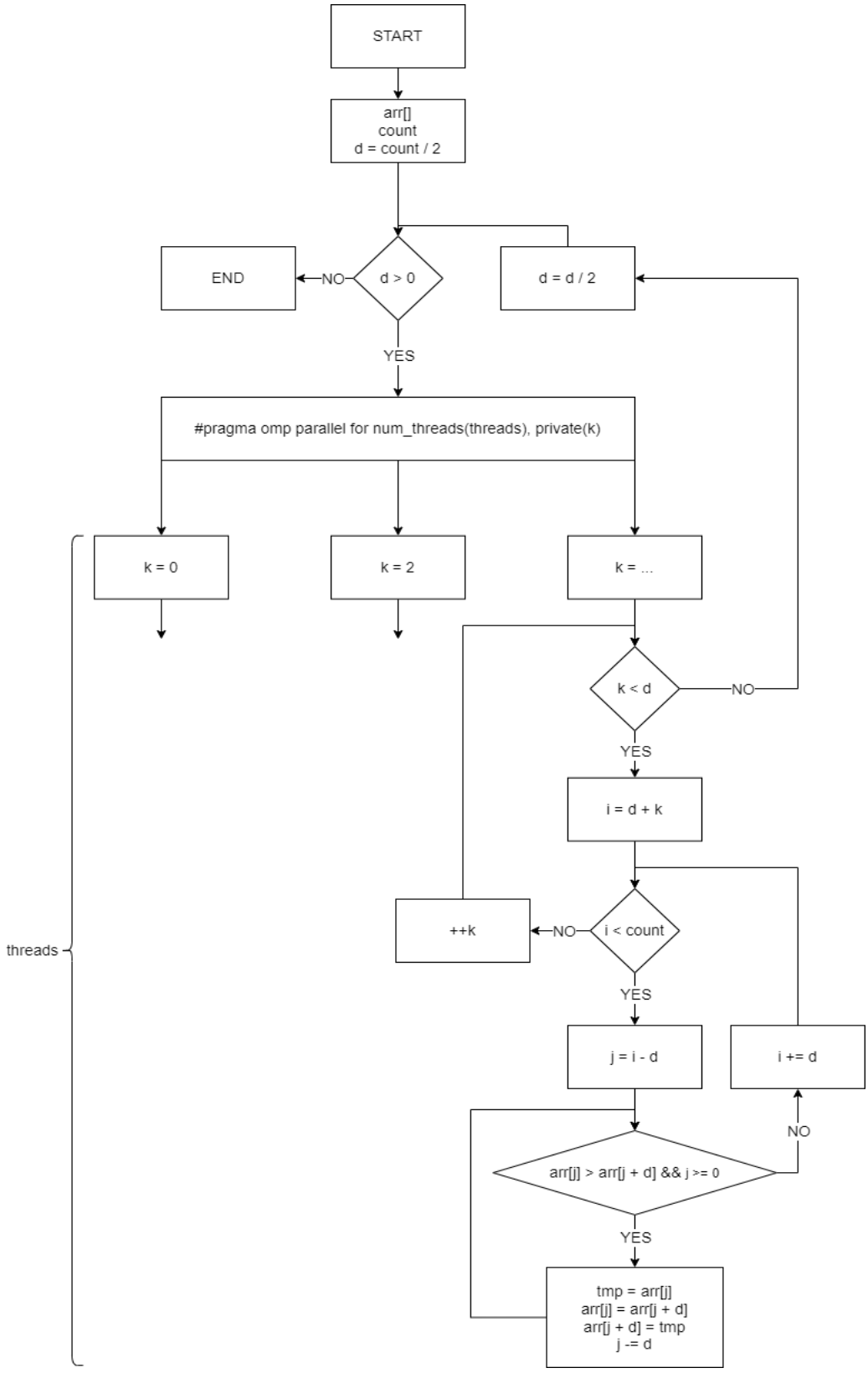
Параллельный алгоритм

Параллельный алгоритм отличается от последовательного распараллеливанием цикла, в котором в разных итерациях нет пересечения сортируемых данных. В данной работе параллельно выполняется цикл, в котором сортируются маленькие массивы с одинаковым числом d, например, первый поток сортирует элементы с индексами: 0, d, 2d, …; второй – с индексами: 1, d+1, 2d+1, …; и т. д.

Блок-схема последовательного алгоритма



Блок-схема параллельного алгоритма



OpenMP

```
12      #pragma omp parallel for num_threads(threads), private(k)
```

- **#pragma omp parallel for** – показывает, что следующий цикл **for** будет выполняться параллельно несколькими потоками;
- **num_threads(threads)** – задает количество потоков, равное **threads**;
- **private(k)** – показывает, что переменная **k** является частной для каждого потока (после завершения параллельного участка считается неопределенной). Если данный параметр не будет указан, то переменная **k** будет иметь правила по умолчанию (в данном случае она будет считаться общей, так как была объявлена вне параллельного блока).

```
9       start = omp_get_wtime();
```

- **omp_get_wtime()** – возвращает астрономическое время прошедшее с определенного момента в прошлом (необходимо: #include <omp.h>.

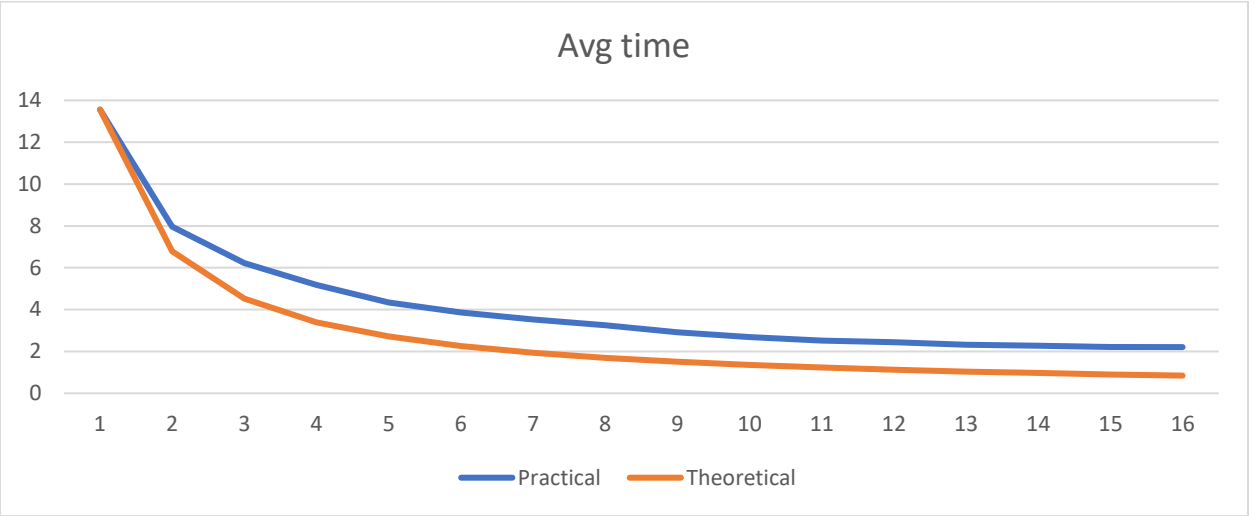
```
18      int available_threads = omp_get_num_procs();
```

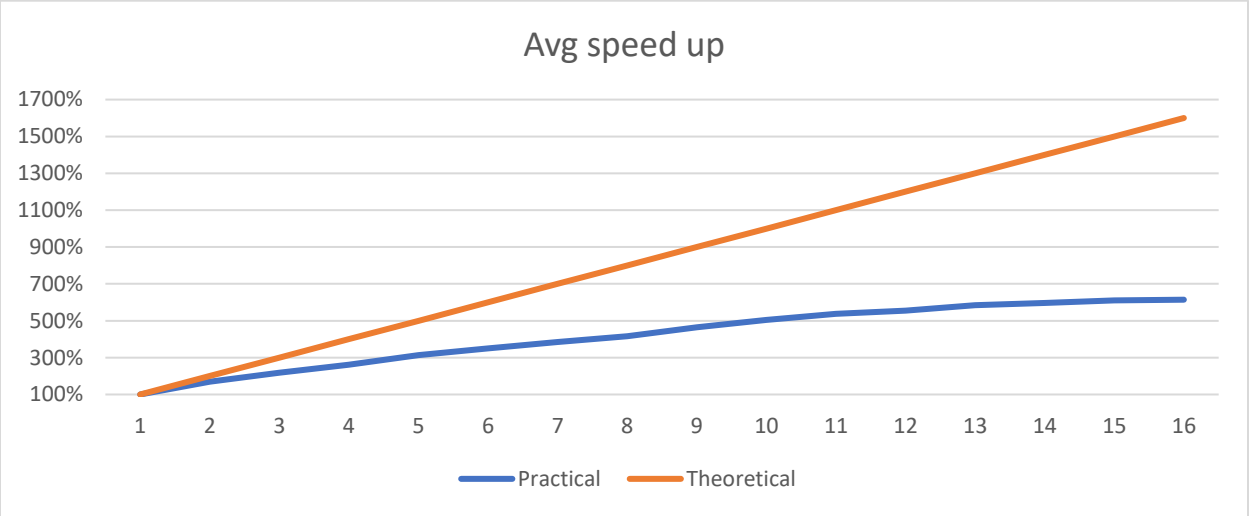
- **omp_get_num_procs() –** возвращает количество доступных процессоров (необходимо: #include <omp.h>).

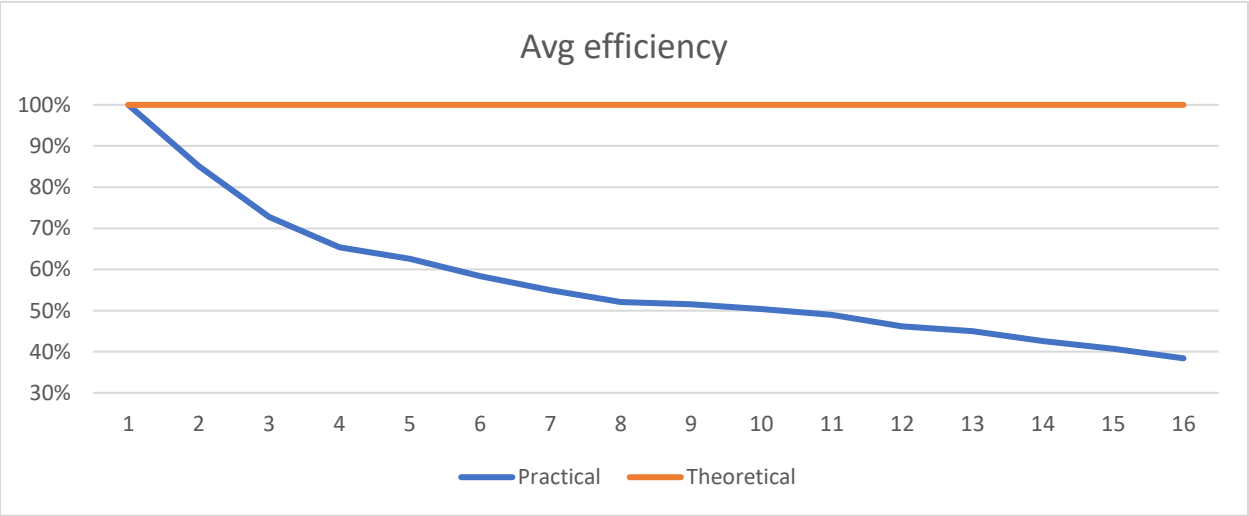
Таблица данных

Threads	Avg time (pr)	Avg time (th)	Avg speed up (pr)	Avg speed up (th)	Avg efficiency (pr)	Avg efficiency (th)
1	13.563884	13.563884	100.00%	100.00%	100.00%	100.00%
2	7.968107	6.781942	170.23%	200.00%	85.11%	100.00%
3	6.214499	4.521295	218.26%	300.00%	72.75%	100.00%
4	5.187171	3.390971	261.49%	400.00%	65.37%	100.00%
5	4.332883	2.712777	313.05%	500.00%	62.61%	100.00%
6	3.872337	2.260647	350.28%	600.00%	58.38%	100.00%
7	3.524715	1.937698	384.82%	700.00%	54.97%	100.00%
8	3.257100	1.695486	416.44%	800.00%	52.06%	100.00%
9	2.923487	1.507098	463.96%	900.00%	51.55%	100.00%
10	2.691758	1.356388	503.90%	1000.00%	50.39%	100.00%
11	2.519715	1.233080	538.31%	1100.00%	48.94%	100.00%
12	2.446818	1.130324	554.35%	1200.00%	46.20%	100.00%
13	2.320515	1.043376	584.52%	1300.00%	44.96%	100.00%
14	2.275720	0.968849	596.03%	1400.00%	42.57%	100.00%
15	2.219406	0.904259	611.15%	1500.00%	40.74%	100.00%
16	2.208289	0.847743	614.23%	1600.00%	38.39%	100.00%

Графики


Avg time
14
12
10
8
6
4
2
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Practical
Theoretical


Avg speed up
1700%
1500%
1300%
1100%
900%
700%
500%
300%
100%
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Practical
Theoretical


Avg efficiency
100%
90%
80%
70%
60%
50%
40%
30%
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Practical
Theoretical

Заключение

Был найден и проанализирован алгоритм сортировки Шелла целочисленного массива (блок-схема). Была разработана и реализована при помощи директив OpenMP параллельная версия алгоритма (блок-схема).

Для массивов псевдослучайных чисел (array[]) были замерены и вычислены временные характеристики работы параллельного алгоритма на различном количестве потоков: среднее время работы, среднее ускорение и средняя эффективность алгоритма.

Были построены графики соотношения теоретических значений характеристик в сравнении с практическими.

Приложение

- https://github.com/KATEHOK/par_prog-3/blob/main/diagram/algorithm.png – блок-схема последовательного алгоритма;
- https://github.com/KATEHOK/par_prog-3/blob/main/diagram/parallel.png – блок-схема параллельного алгоритма;
- https://github.com/KATEHOK/par_prog-3/blob/main/src/main.c – исходный код программы;
- https://github.com/KATEHOK/par_prog-3/blob/main/src/shell_sort.h – файл заголовков;
- https://github.com/KATEHOK/par_prog-3/blob/main/src/shell_sort.c – исходный код алгоритма;
- https://github.com/KATEHOK/par_prog-3/blob/main/report/data.txt – «сырые» данные (текст);
- https://github.com/KATEHOK/par_prog-3/blob/main/report/lab3.xlsx – «обработанные» данные (таблица и графики);
- https://github.com/KATEHOK/par_prog-3/blob/main/report/lab3.docx – отчет (docx);
- https://github.com/KATEHOK/par_prog-3/blob/main/report/lab3.pdf – отчет (pdf).